# 第 1 章

# 认识 JMeter

随着软件系统的发展，性能测试在软件开发行业中被日益重视，同时也出现了专业的性能测试工具来代替人工性能测试。JMeter 正是在这样的背景下应运而生。JMeter 是早期出现的性能测试工具之一，一经推出之后就一直深受测试工程师的喜爱。因为有了 JMeter 后，软件的性能测试变得更加简单，同时提高了测试工程师的工作效率。本章将带领读者认识 JMeter 这款性能测试工具，并介绍性能测试以及性能测试工具的发展历程。学习完本章后，读者会对性能测试工具以及 JMeter 有一个初步的认识。

## 1.1　JMeter 基本介绍

JMeter 是 Apache 基金会提供的一个开源的、由纯 Java 语言编写的性能测试工具，最初仅被设计用于 Web 应用测试，后来随着性能测试等其他测试类型的出现，才被逐步扩展到了其他测试领域中。我们可以通过访问 JMeter 官方网站了解其技术信息，如图 1-1 所示。

JMeter 的主要特点如下：

（1）完全开放源码，并且所有的功能都是免费的，用户也可以免费使用和修改源码以满足特定的性能测试需求。

（2）支持众多网络层/应用层的通信协议（比如 TCP、HTTP、FTP、JDBC、SMTP、POP3、IMAP、JMS 等），JMeter 几乎可以支持对所有的应用系统进行性能测试。

（3）JMeter 支持插件和扩展，可以扩展其功能和性能，以满足特定的性能测试需求。

（4）JMeter 完全可移植，且由纯 Java 语言编写，因此可以兼容不同的操作系统。

（5）支持定制性能测试场景，比如设置并发用户数、持续时间、循环次数和延迟时间等，以模拟真实的使用场景。

（6）支持聚合报告、图形结果、树形结果等测试结果收集和显示方式，便于性能分析和调优。

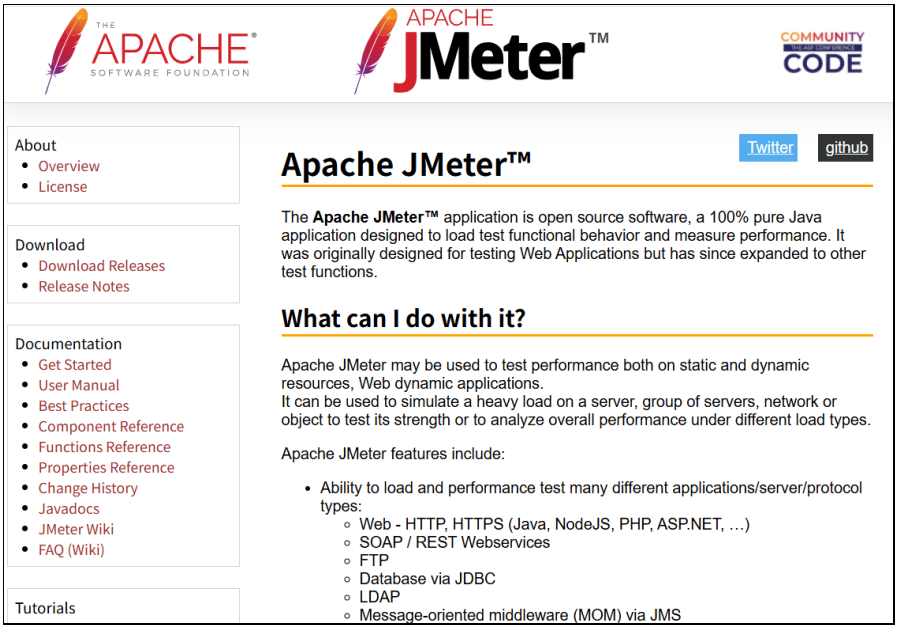


图 1-1　JMeter 官方网站

JMeter 的源码托管在 GitHub，通过 GitHub 可以访问 JMeter 的源码托管界面，如图 1-2 所示。

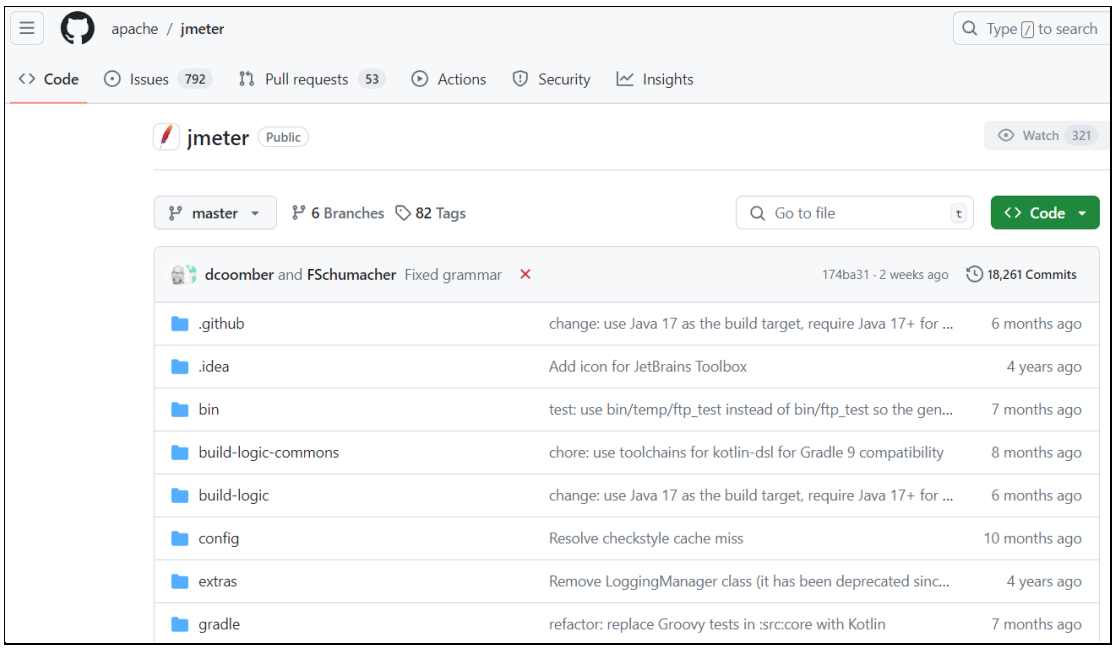


图 1-2　JMeter GitHub 网站

从图 1-2 中可以看到，JMeter 当前有超过 18261 次的源码提交记录，并且在开源社区拥有众多的代码贡献者，可见 JMeter 在开源社区中保持着相当大的活跃度。我们从中可以看到性能测试的重要性，也能看到 JMeter 因其开源免费、技术资料众多、社区庞大的优点被广大性能测试爱好者和工作者喜爱。同时，在 JMeter 的 GitHub 网页中，还介绍了如何参与 JMeter 的源码贡献。JMeter 鼓励性能测试爱好者参与 JMeter 的源码贡献开发。

JMeter 于 1998 年 12 月 15 日发行了第一个版本 1.0，之后一直保持着非常高的更新频率，当前 JMeter 官网的最新版本为 5.6.3。JMeter 的所有历史版本变更记录如图 1-3 所示。这对于一个性能测试工具来说是非常不容易的，因为在过去的近 30 年，这款性能测试工具一直在进行更新和维护。

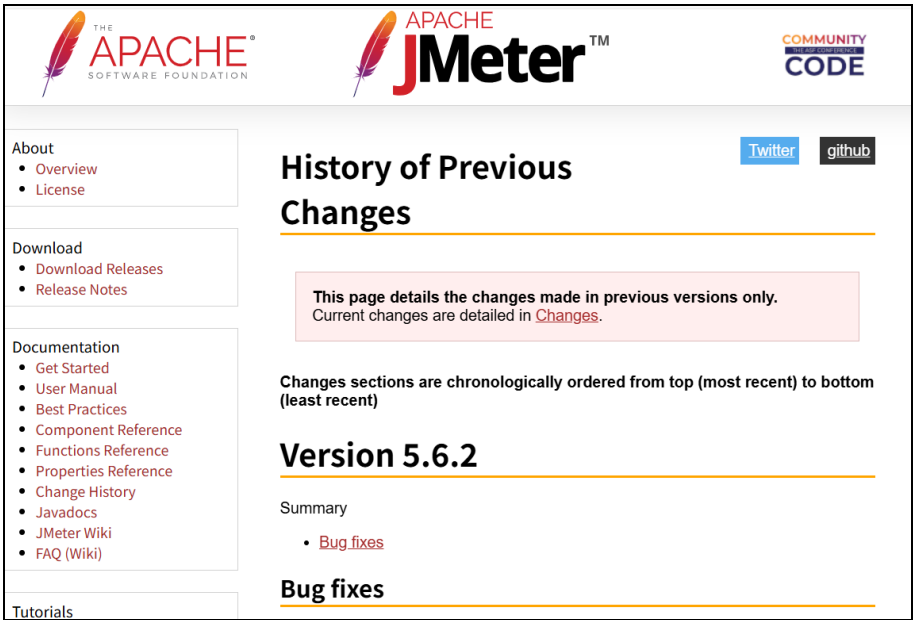


图 1-3　JMeter 历史版本变更记录

## 1.2　性能测试的发展

JMeter 的出现与性能测试的发展是密不可分的。性能测试最早可以追溯到 20 世纪 70 年代，当时的性能测试仅针对硬件设备，但随着计算机技术突飞猛进地发展，大量商务网站的出现使得越来越多的人开始关注软件系统在性能上的表现，性能测试也开始被广泛应用于电子商务网站等多个领域，使得性能测试成为软件开发生命周期中非常重要的环节之一。其实在软件开发的最早时期，软件开发生命周期中并没有性能测试这个专门的测试类型，当时性能测试是由开发工程师来完成的，随着商业化性能测试软件 LoadRunner 的出现，使得性能测试逐步由测试工程师来主导完成。

随着软件系统日益复杂以及软件系统规模的不断扩大，性能测试开始逐步与功能测试进行

完美结合，用于对系统性能进行评估。但为了更准确地模拟真实使用场景以及关注系统的整体稳定性，负载测试和压力测试也开始慢慢被引入性能测试中，作为性能测试的子项来丰富其种类。负载测试通常用于评估软件系统在不同的负载下的性能表现，而压力测试通常用于模拟和评估系统在超高负载下的稳定性和可靠性。

性能测试经历了从硬件领域到软件领域的过渡。在互联网、大数据、人工智能高速发展的当今社会，性能已经成为用户能直接体验到的最重要的一个环节。因此，性能测试变得越来越重要，它能够帮助产品经理以及开发工程师评估系统在不同负载下的性能表现，同时为代码优化、架构优化和系统调整提供性能依据。

对于性能测试工程师来说，需要持续学习性能测试工具的使用以及性能分析、诊断、调优的技能，以便能更好地适应性能测试的发展和创新。对于企业或者组织来说，需要更加重视软件的性能测试，以便让用户拥有更好的性能体验。

## 1.3 性能测试工具的发展

性能测试工具的发展主要包括如下几个阶段：

1）开发工程师测试阶段

由于早期没有专业的性能测试工具，也没有专门的性能测试工程师，因此性能测试主要依靠开发工程师自己通过编写测试代码来完成，而且这个阶段的性能测试也比较简单，几乎没有相关的体系和规范。

2）性能测试工具的初期阶段

在 20 世纪末，随着软件系统的发展，性能测试被日益重视。也正是在这个阶段，开始出现了专业的性能测试工具，比如 1998 年 JMeter 发布了第一个版本，1999 年，LoadRunner 也发布了第一个版本。但是早期的性能测试工具都比较简单，比如早期的 LoadRunner 仅用于模拟多个用户同时访问某个软件系统，并且仅能收集一些性能测试数据以用于评估系统的性能表现。此时的性能测试工具的功能都比较少，无法满足一些复杂业务场景的性能测试需求，并且无法提供更多的性能分析诊断功能。这个阶段开始出现了一些简单的性能测试体系和规范，但远远不够完善。

3）性能测试工具的发展阶段

LoadRunner 在发布了第一个版本后受到了大量软件开发者的好评，随后又发布了第二个版本，在这个版本中开始支持 Web 协议、数据库协议等网络应用协议，并且可以模拟多种 Web 浏览器的行为（比如单击按钮、填写表单等）。2003 年，LoadRunner 发布了第三个版本，增加了对移动应用程序、不同网络环境的模拟等功能的支持。与此同时，JMeter 也在这个阶段进行了大量的版本迭代和发布，仅在 1999 年，JMeter 就发布了超过 4 个版本，支持了那个时期常见的 HTTP 等 Web 协议的性能测试。

4）性能测试工具的井喷阶段

在这个阶段，除了 LoadRunner、JMeter 在不断地迭代发布之外，还出现了以下测试工具：

- WebLoad。
- NeoLoad。
- Gatling。
- Ngrinder。

这些性能测试工具各有特点，它们不仅推动了性能测试工具的发展，还极大地推动了性能测试这个行业的发展。

5）云服务性能测试工具阶段

随着云计算的出现和高速发展，软件系统的部署不再局限于本地部署或者自建机房部署，而是更多地在云上部署。因此，很多云服务的厂商也趁机推出了自己的云服务性能测试工具，比如阿里云推出了性能测试工具 PTS。当然，云服务性能测试也可以通过传统的 JMeter、LoadRunner 等工具来完成。

6）性能测试工具的未来阶段

随着人工智能（AI）的快速发展，相信未来的性能测试工具也会更加 AI 化，可能不再需要性能测试工程师编写性能测试脚本，使得性能测试变得更加简单。

## 1.4　选择 JMeter 的原因

性能测试发展到现在已经非常成熟，而且可选的性能测试工具也很多。本节将对常见的性能测试工具做一个对比，帮助读者了解这些软件的特点，具体如表 1-1 所示。

**表 1-1　性能测试工具的优缺点**

| 性能测试工具 | 优　　点 | 缺　　点 |
| --- | --- | --- |
| JMeter | （1）开源免费，几乎支持对所有的软件系统做性能测试，而且支持插件和扩展，可以扩展其功能和性能以满足特定的性能测试需求。<br>（2）支持性能测试流程编排，并且也支持断言、逻辑控制器等高级性能测试逻辑，可满足复杂的性能压测需求。<br>（3）支持分布式部署，可以模拟大量的高并发请求。<br>（4）学习成本低，相关的技术资料非常齐全，社区非常强大 | （1）使用 Java 语言开发，软件界面功能比较简单，体验性较差。<br>（2）可查看的监控和报告指标较少。<br>（3）分布式部署时较复杂，维护和管理分布式集群的成本较大 |

（续表）

| 性能测试工具 | 优　　点 | 缺　　点 |
| --- | --- | --- |
| LoadRunner | （1）商业付费软件，拥有可靠的售后支持，在国内的知名度很高。<br>（2）提供非常强大的负载测试能力，支持分布式部署，能生成详细的性能测试结果和报告。<br>（3）成熟、稳定的企业级性能测试，适用于超大规模性能并发测试，并且提供丰富的测试场景和性能结果分析功能。<br>（4）支持 Web 协议的性能测试脚本录制，可以减少编写性能测试脚本的难度 | （1）付费软件，购买价格较高，软件安装包较大，并且安装程序繁杂。<br>（2）操作难度较大，需要一定的学习和培训成本。<br>（3）底层基于 C 语言开发，在编写性能测试脚本时，需要对 C 语言脚本有一定的了解。<br>（4）只能运行在 Windows 系统上，不能兼容其他的操作系统 |
| WebLoad | （1）支持超高并发的性能压测，并且兼容支持包括 Windows、Solaris 和 Linux 在内的众多操作系统。<br>（2）性能测试报告提供了详细和深入的性能分析数据，帮助性能测试工程师更好地理解系统的性能表现 | （1）脚本语言是不常见的标准语言，学习起来较为困难，学习成本较高。<br>（2）商业付费软件，购买价格较高，虽然也提供社区版，但是社区版只支持单机模式 |
| NeoLoad | （1）提供负载测试、压力测试、稳定性测试、容量规划等多种功能，以满足不同测试需要。<br>（2）支持包括 HTTP、HTTPS、SOAP、REST、JDBC、JMS、FTP 等在内的多种应用协议。<br>（3）提供了丰富的报告功能，可以直观地展示测试结果和性能指标，方便分析和优化性能问题 | （1）商业付费软件，购买价格较高。<br>（2）对于初学者来说，需要一定的学习成本才能熟练掌握其测试功能和操作方式 |
| Gatling | （1）支持复杂的场景编排，能够模拟各种用户行为和业务场景。<br>（2）采用了异步非阻塞的 I/O 模型（Akka 架构），可以支持高并发的性能测试，性能表现非常出色。<br>（3）开源，可以免费使用 | （1）需要一定的编程基础，对于没有任何编程经验的用户来说，可能需要较高的学习成本。<br>（2）性能测试报告较弱，但是可以通过扩展组件来获取更多的性能监控信息。<br>（3）主要支持 Web 应用程序的性能测试，对于其他类型的应用程序支持不友好 |
| Ngrinder | （1）采用 Web 界面来管理性能测试脚本和进行性能测试，以及查看测试报告，使用较为简单。<br>（2）支持分布式性能压测。在运行分布式压测时，系统由一个 controller 和连接它的多个 agent 组成，controller 会把测试分发到多个 agent 上去执行。用户可以设置使用多个进程和线程来并发地执行性能测试脚本，而且在同一线程中，通过重复不断地执行对应的性能测试脚本，来模拟多个并发用户执行 | （1）性能测试脚本基于 Python 语言编写，对于一些复杂场景的性能测试，需要测试人员对 Python 有一定认识。<br>（2）测试报告和监控界面较为简单，无法获取到更多的、详细的监控数据 |

从对比的情况看，JMeter 是一个适合中小型软件公司使用的性能测试工具，因为 JMeter 完

全开源和免费，并且在强大的社区支持下一直保持着活跃的版本更新。JMeter 也是一款非常适合普通性能测试工程师学习的、优秀的性能测试工具，因为在互联网和社区中可以搜索到非常多的 JMeter 技术资料，在遇到问题时很容易找到相关资料并快速地解决问题。

## 1.5　JMeter 的安装和部署

前面已经提到 JMeter 是完全由 Java 语言来实现的一款性能测试工具，而 Java 语言又拥有非常好的平台兼容性，所以 JMeter 可以在不同类型的操作系统上运行。本节以在 Windows 操作系统上安装 JMeter 为例，讲解 JMeter 的安装和部署方法。具体的步骤如下：

步骤 01　从 JDK（Java Development Kit，Java 开发工具）官方网站中下载 JDK。安装 JMeter 对 JDK 的要求是不能低于 JDK 8 版本，通常建议使用 JDK 11 版本，如图 1-4 所示。读者可以根据自己的操作系统类型选择对应的版本进行下载。

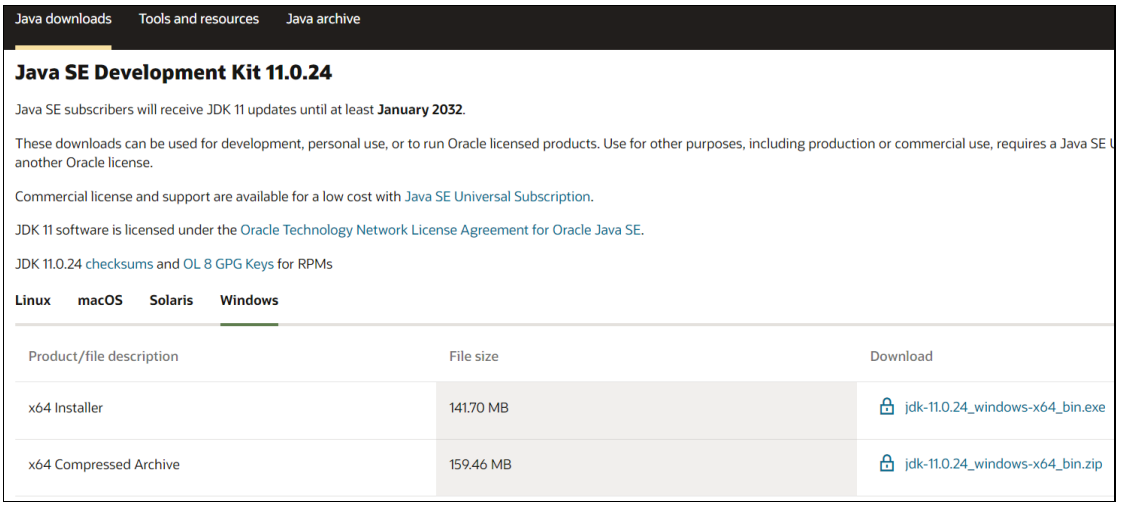


图 1-4　JDK 官方网站

步骤 02　双击后缀名为 exe 的安装文件，按照界面上的提示进行安装即可。安装好的目录如图 1-5 所示。

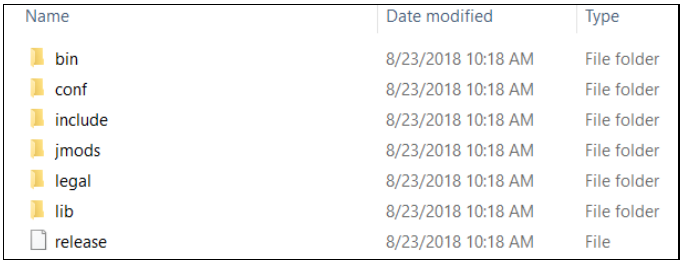


图 1-5　解压后的 JDK 目录

步骤 03 在操作系统中配置环境变量，如图 1-6 所示，新建一个 JAVA_HOME 的变量，填入 JDK 安装或者解压后的文件夹路径，然后编辑 Path 变量，在其值后输入%JAVA_HOME%\bin。

步骤 04 从 JMeter 官方网站中下载最新版本的 JMeter 安装包，目前的最新版本是 5.6.3，如图 1-7 所示。在 Binaries 中有两种下载格式，tgz 和 zip，随意选择一种下载即可。


图 1-6 配置环境变量

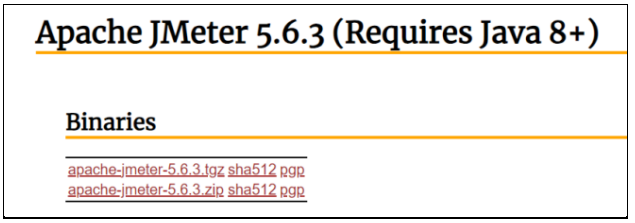


图 1-7 JMeter 安装包下载

步骤 05 将下载好的 JMeter 安装包解压到操作系统下的一个文件夹中，如图 1-8 所示。

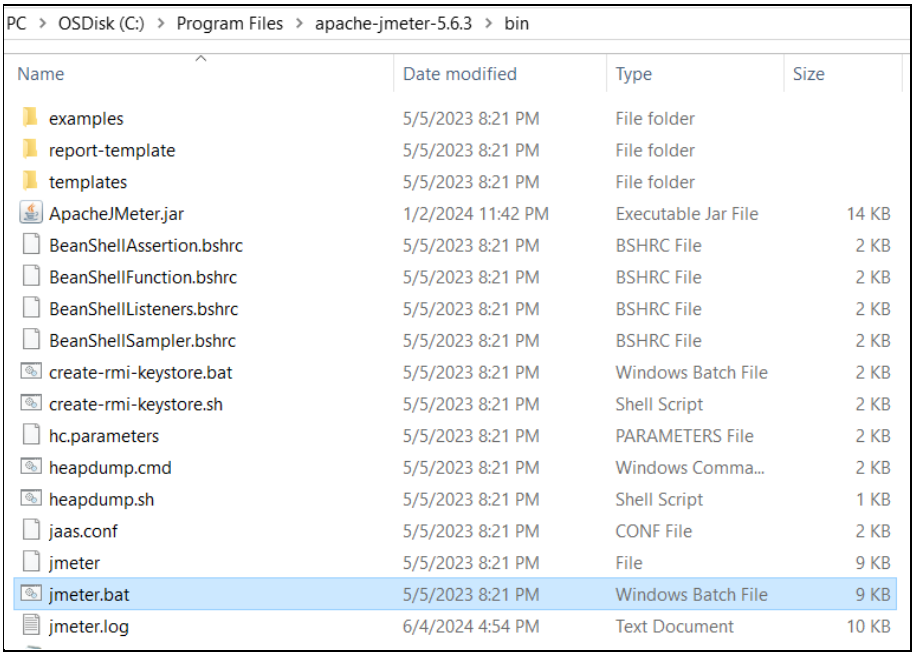


图 1-8 解压后的 JMeter 目录

步骤 06 双击其中的 jmeter.bat 文件，即可启动 JMeter 并进入 JMeter 的图形运行界面，如图 1-9 所示。

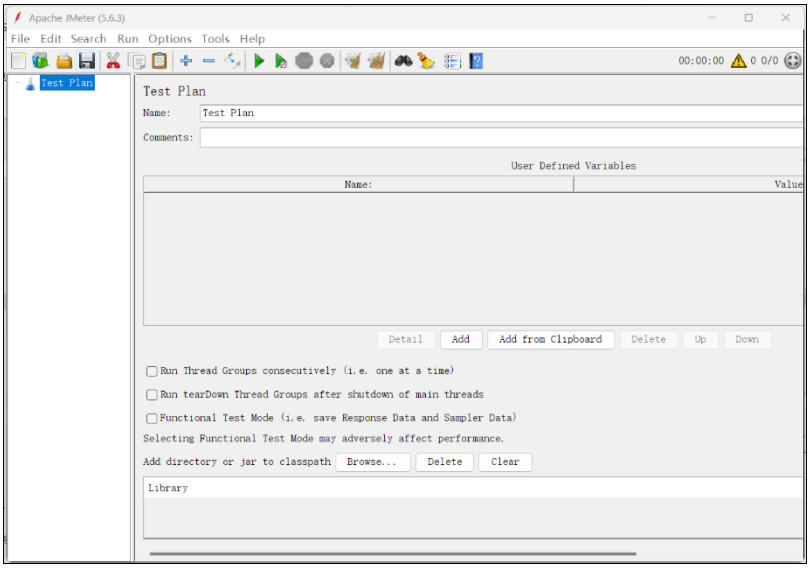


图 1-9　启动后的 JMeter 运行界面

## 1.6　JMeter 的元件

元件是 JMeter 的操作模块，通常一个完整的性能测试需要由部分元件或者全部元件串联起来，合作完成性能测试脚本的编写。本节将对 JMeter 的 9 个常用元件（也称组件）做初步介绍。

1）线程组（Threads Group）

主要用于控制整个性能测试的运行开始时间以及运行时长、线程数量（即并发用户数）等。通常情况下，每个性能测试场景都需要先在 JMeter 界面上创建一个线程组，然后才能运行后续的性能测试脚本。这是因为性能压测通常需要模拟大量的用户来进行并发操作，所以需要通过线程组使用多个线程的方式来模拟产生大量的用户。

2）配置元件（Config Element）

主要用于完成性能测试中需要的一些配置信息，包括初始化变量或者参数的默认值、读取 CSV 文件数据、设置公共请求参数、赋予变量值等，以便后续取样器（Sampler）直接使用。通常，性能测试脚本中变量的参数化也可以通过配置元件来实现，这在后续的章节中会进行详细讲解。

3）前置处理器（Pre Processors）

即预处理器，用于在实际取样器发出请求之前，对即将发出的请求进行初始化的预处理。

4）取样器（Sampler）

通常用来模拟并发用户的操作，向待性能压测的对象发送请求以及接收相应的响应数据。取样器是 JMeter 性能压测的核心元件，通常情况下，要完成一个性能测试场景，那么肯定是离

不开取样器的。

5）逻辑控制器（Logic Controller）

通常用来控制采样器的执行顺序，同时也可以对 JMeter 中元件的执行逻辑进行控制。因为在做性能测试时，经常会遇到比较复杂的业务场景，就可以使用逻辑控制器来完成一些特定的、比较复杂的业务逻辑处理。

6）后前置处理器（Post Processors）

用于在实际取样器发出请求之后对请求的响应结果进行后置处理。

7）断言（Assertions）

通常用于对取样器返回的结果做检查，以判断返回的响应结果是否正确，进而判断某次性能测试的结果是否通过，等同于 LoadRunner 中的检查点功能。

8）监听器（Listener）

通常用来监听及展示 JMeter 取样器的执行结果。监听器支持以树、表及图形等形式展示当前性能压测的结果，也可以用文件的方式保存测试结果。JMeter 支持用 XML、CSV 等格式的文件来保存测试结果。监听器通常用于对性能测试的结果进行统计分析，以便快速发现性能压测中可能存在的性能问题。

9）定时器（Timer）

类似于 LoadRunner 中的思考时间（think time），用来设置线程的延迟和同步时间，通常在每个取样器发出请求之前执行。

这些元件在启动 JMeter 后，可以通过如图 1-10 所示的方式来添加。

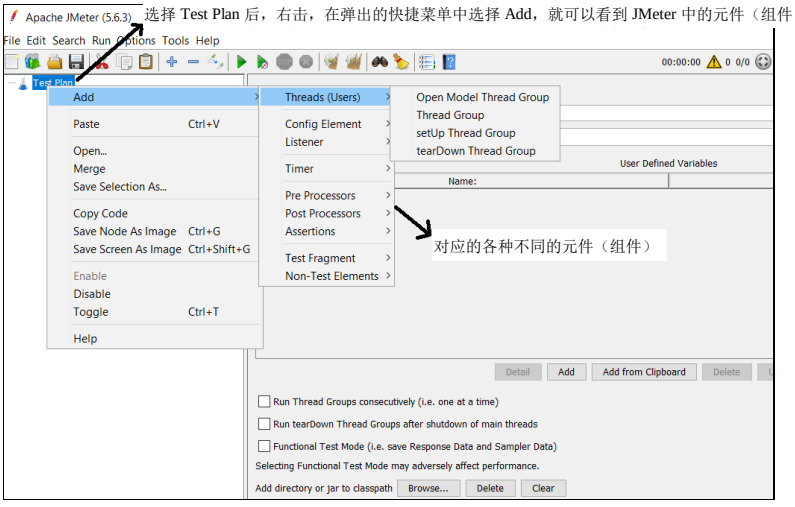


图 1-10　添加 JMeter 元件

JMeter 界面默认显示的文字是英文，我们可以通过依次单击界面上的 Options→Choose Lanuage 菜单来切换为中文显示，如图 1-11 所示。

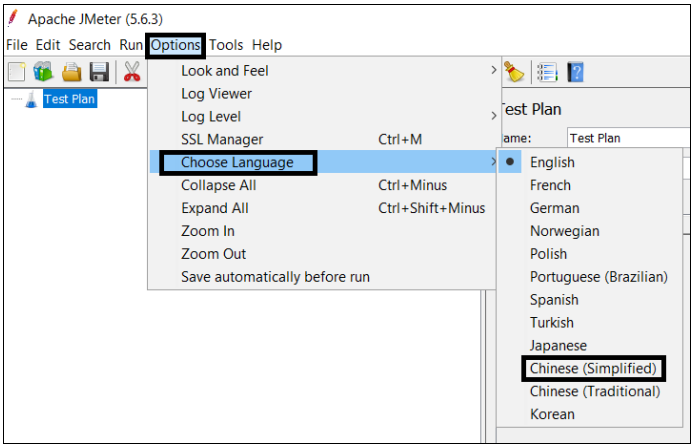


图 1-11　切换为中文显示

切换完成后，就可以看到如图 1-12 所示的中文界面了。中文界面可以让我们更容易理解 JMeter 提供的菜单以及相关的功能。

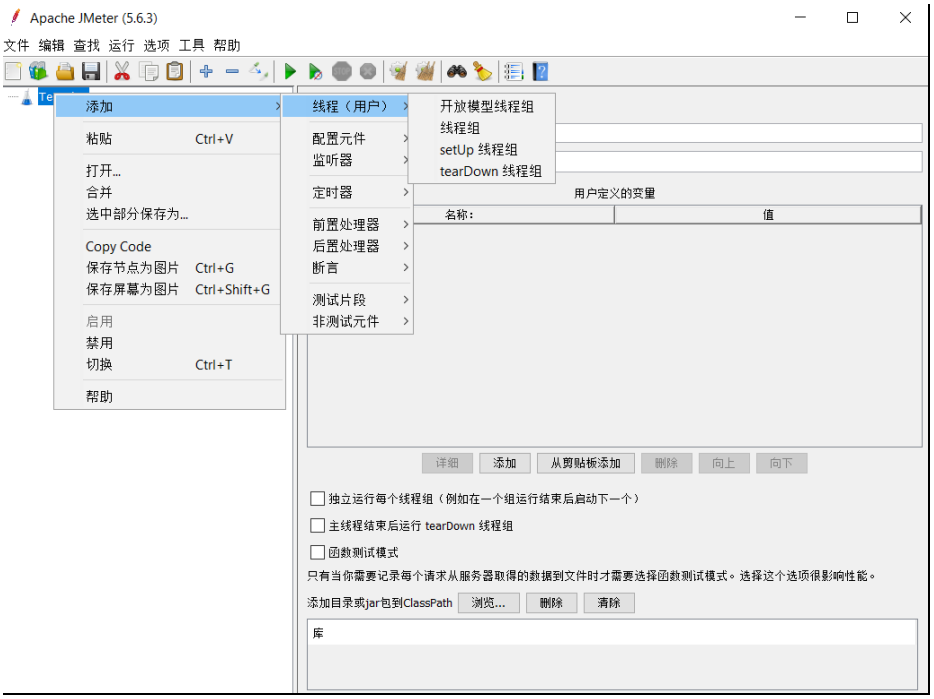


图 1-12　JMeter 中文界面

为了方便读者学习 JMeter，在本书的后续章节中，都将使用中文界面来对 JMeter 进行深入讲解。

使用 JMeter 开始一个性能测试的主要步骤如图 1-13 所示。通常情况下，线程组、取样器、断言、监听器是性能测试必须用到的元件。对于一些复杂的业务场景，可能需要用到更多的元件。

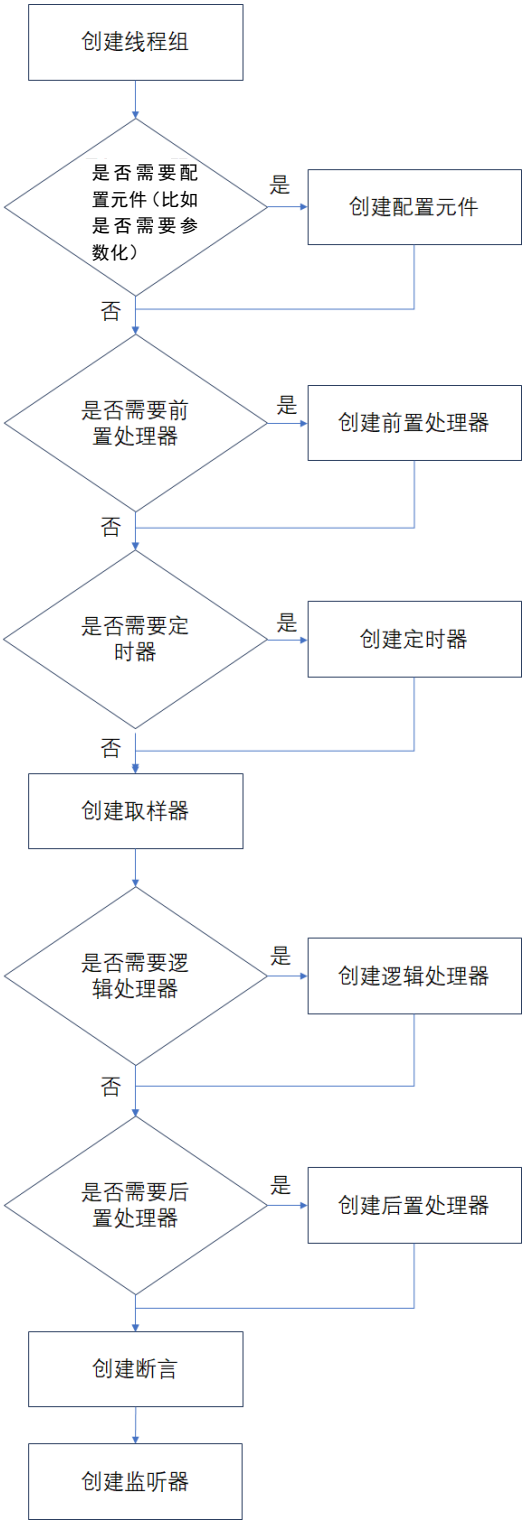


图 1-13　JMeter 性能测试的主要步骤

# 1.7　JMeter 的运行模式

JMeter 常用的运行模式主要包括 GUI 模式、命令行模式以及服务器模式。大多数场景下，直接运行 jmeter.bat 或者 jmeter.sh 启动的就是 GUI 模式。jmeter.bat 通常是在 Windows 操作系统下启动 GUI 模式，而 jmeter.sh 通常是在 Linux 或者 UNIX 操作系统下启动 GUI 模式。GUI 模式通常用于 JMeter 性能测试脚本的编写，命令行模式通常用于性能压测，而服务器模式则用于分布式压测。

## 1.7.1　GUI 模式

GUI 模式会启动 JMeter 的标准界面，JMeter 的使用者可以通过界面来进行操作，这样更加直观和简单。在 jmeter.bat 批处理文件中，定义了启动 GUI 模式的具体命令，主要包括 JDK、JVM（Java Virtual Machine，Java 虚拟机）参数，以及运行 JMeter 所需的 JAR 包。GUI 模式启动的过程如图 1-14 所示。

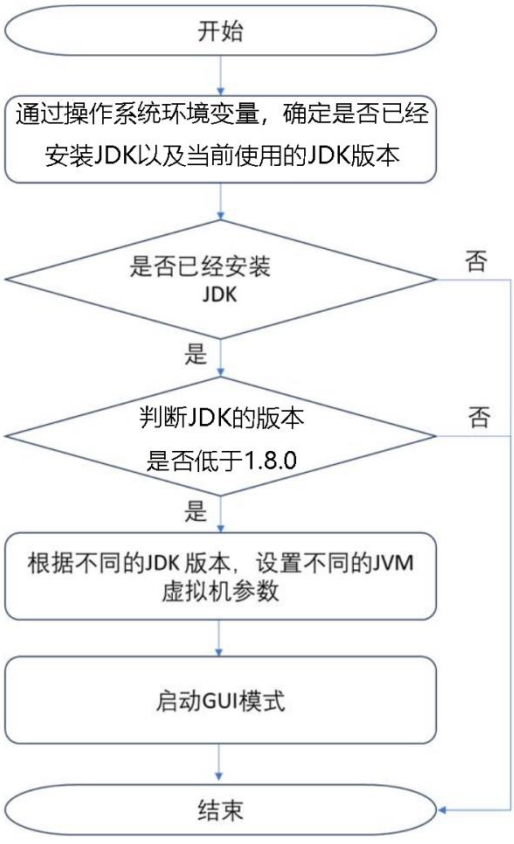


图 1-14　JMeter GUI 模式的启动过程

## 1.7.2 命令行模式

因为 JMeter 是 Java 语言开发的，而对于 Java 语言来说，GUI 界面并不是其擅长的，在高并发压测时，GUI 界面通常容易卡死，所以在 GUI 模式中完成性能测试脚本的编写后，建议在实际运行压测时，切换到命令行模式。命令行模式顾名思义就是在 Windows 或者 Linux、UNIX 操作系统中，直接通过运行 JMeter 命令来执行性能压测。命令行模式支持的命令行参数如下：

（1）-h, --help：输出命令行的所有帮助信息。

（2）-v, --version：输出 JMeter 的版本信息。

（3）-p, --propfile <argument>：启动时，指定 JMeter 的属性配置文件的路径。

（4）-q, --addprop <argument>：启动时，添加额外的 JMeter 的属性配置文件的路径。

（5）-t, --testfile <argument>：启动时，指定要运行的 JMeter 的测试计划的文件（.jmx）。在 GUI 模式下编写好的性能测试计划脚本，可以保存为.jmx 文件。

（6）-l, --logfile <argument>：启动时，指定 JMeter 取样器日志的输出路径。

（7）-i, --jmeterlogconf <argument>：启动时，指定 JMeter 的日志配置文件的路径。默认会使用 bin 目录下的 log4j2.xml。

（8）-j, --jmeterlogfile <argument>：启动时，指定 JMeter 日志的输出路径。默认会输出到 bin 目录下的 jmeter.log 中。

（9）-n, --nongui：启动时，以无 GUI 界面的模式运行，也就是启动时不会展示 JMeter 的 GUI 界面。

（10）-s, --server：以 JMeter 服务器的模式运行。

（11）-E, --proxyScheme <argument>：启动时，增加设置代理服务器。

（12）-H, --proxyHost <argument>：启动时，设置使用的代理服务器的 IP 地址或者域名。

（13）-P, --proxyPort <argument>：启动时，设置使用的代理服务器的端口号。

（14）-N, --nonProxyHosts <argument>：启动时，设置哪些 IP 地址或者域名不需要代理服务器进行代理。

（15）-u, --username <argument>：启动时，设置代理服务器的用户名。

（16）-a, --password <argument>：启动时，设置代理服务器的密码。

（17）-J, --jmeterproperty <argument>=<value>：启动时，以 key-value（键−值）的形式指定需要添加的 JMeter 属性配置。

（18）-G, --globalproperty <argument>=<value>：启动时，以 key-value 的形式指定需要添加的 JMeter 全局属性配置。

（19）-D, --systemproperty <argument>=<value>：启动时，以 key-value 的形式指定需要添加的系统属性配置。

（20）-S, --systemPropertyFile <argument>：启动时添加额外的系统属性配置文件的路径。

（21）-f, --forceDeleteResultFile：设置是否强制删除现有的已经存在的性能测试结果文件和 Web 报告文件夹。

（22）-L, --loglevel <argument>=<value>：设置指定组件的日志输出级别。比如，可以设置 jmeter.util=DEBUG，因为 jmeter.util 是 JMeter 中的一个组件名。

（23）-r, --runremote：启动时，设置需要启动的远程服务器的 IP 地址或者域名地址，这些要启动的远程服务器地址需要在 remote_hosts 属性中配置。

（24）-R, --remotestart <argument>：启动时，设置需要启动的远程服务器的 IP 地址或者域名地址。如果使用了该参数，将会覆盖 remote_hosts 属性配置中配置的、需要远程启动的服务器地址。

（25）-d, --homedir <argument>：设置 JMeter 运行时的主目录路径。

（26）-X, --remoteexit：设置测试结束时退出远程服务器，但是当前自己所在的 GUI 模式的界面不会退出。

（27）-g, --reportonly <argument>：设置性能测试时，仅通过测试结果文件来生成最终的测试报告仪表板。

（28）-e, --reportatendofloadtests：设置性能测试时，在负载测试结束后，生成测试报告仪表板。

（29）-o, --reportoutputfolder <argument>：设置性能测试时，生成的测试结果报告的输出文件夹路径。

## 1.7.3　服务器模式

在性能测试中，当一台 JMeter 压测机无法提供足够的并发用户时，就需要用多台压测机同时提供并发用户来进行性能压测，如图 1-15 所示。

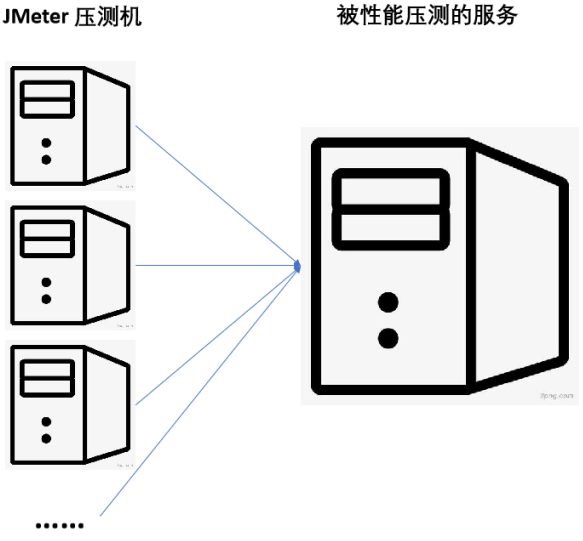


图 1-15　JMeter 服务器集群

当出现这种需要高并发用户场景时，就会用到 JMeter 的服务器模式了。此时 JMeter 会存

在多个工作节点，每个节点都会以服务器的模式启动。由于启动了多个服务器节点，因此肯定需要一个 JMeter 管理节点来管理这些工作节点，如图 1-16 所示。

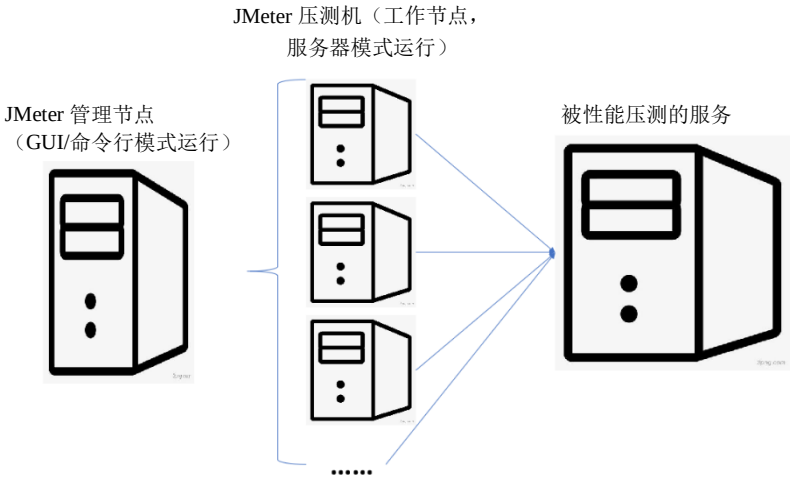


图 1-16　JMeter 管理模式

（1）当需要进行分布式压测时，需要在每一台性能压测机中安装 JMeter，安装方式可以参考 1.5 节中的介绍。

（2）在每台性能压测机中修改 JMeter 的 jmeter.properties 属性配置文件，该文件位于 JMeter 的 bin 目录中。在 jmeter.properties 属性配置文件中，有关远程主机和 RMI 的内容如下：

```
...
#---------------------------------------------------------------------------
# Remote hosts and RMI configuration
#---------------------------------------------------------------------------

# Remote Hosts - comma delimited
remote_hosts=127.0.0.1
#remote_hosts=localhost:1099,localhost:2010

# RMI port to be used by the server (must start rmiregistry with same port)
#server_port=1099

# To change the port to (say) 1234:
# On the server(s)
# - set server_port=1234
# - start rmiregistry with port 1234
# On Windows this can be done by:
# SET SERVER_PORT=1234
# JMETER-SERVER
#
# On Unix:
# SERVER_PORT=1234 jmeter-server
```

```
    #
    # On the client:
    # - set remote_hosts=server:1234

    # Parameter that controls the base for RMI ports used by RemoteSampleListenerImpl
and RemoteThreadsListenerImpl (The Controller)
    # Default value is 0 which means ports are randomly assigned
    # If you specify a base port, JMeter will (at the moment) use the ports that
start one after the given base.
    # You may need to open Firewall port on the Controller machine
    #client.rmi.localport=0
    ...
```

其中，常用的配置项说明如下：

- remote_hosts：在 remote_hosts 中添加每一台 JMeter 工作节点的 IP 地址，以英文逗号隔开。
- server_port：默认为 1099，该端口用于当 JMeter 以服务器模式启动时，启动一个 RMI 服务端口。RMI JDK 在 1.2 版本中实现了一个基于 Java 语言的远程调用方法，RMI 的出现让 Java 语言有了分布式处理的能力。注册监听服务（即 rmiregistry 服务），如图 1-17 所示，当 RMI 通信时，会先连接到该端口上查找当前可用的 Server 端服务。

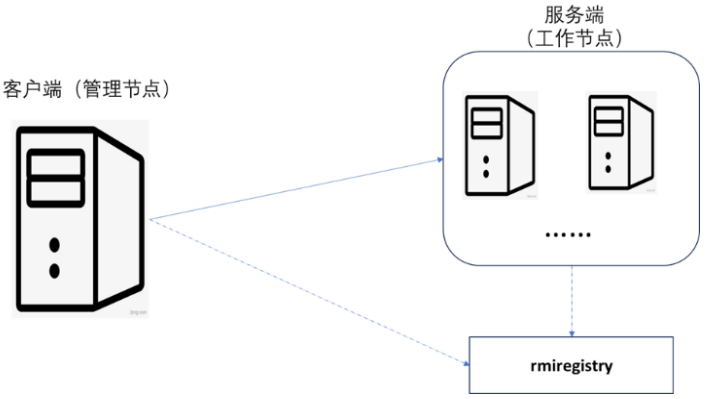


图 1-17　RMI 通信链路

- client.rmi.localport：该端口默认为 0，表示让操作系统随机分配一个可用的端口。该端口用于在和 server_port 端口进行通信时，启动一个 RMI 本地服务通信端口。server_port 和 client.rmi.localport 都用于 RMI 通信。

在完成上述内容的设置后，如果 JMeter 部署在 Windows 操作系统中，可以运行 jmeter-server.bat 来启动 JMeter 的服务器模式；如果 JMeter 部署在 Liunx/UNIX 操作系统中，可以运行 jmeter-server.sh 来启动 JMeter 的服务器模式。

在每一台 JMeter 的工作节点中完成上述配置，并以服务器模式运行 JMeter 后，我们可以在 JMeter 的管理节点上以 GUI 模式启动 JMeter。在 GUI 界面的运行菜单下，可以看到远程启

动服务器的相关信息，如图 1-18 所示。我们可以选择远程启动单台远程服务器来运行当前的性能测试计划，比如选择 server1_ip 来运行；也可以选择菜单上的“远程启动所有”选项，以在所有的远程服务器上运行当前的性能测试计划。


图 1-18　JMeter 远程启动界面

## 1.8　本章总结

本章主要介绍了 JMeter 的基本信息、性能测试的发展历程、JMeter 的安装部署以及 JMeter 的主要组成部分。读者需要掌握以下重点内容：

- 如何在本地安装和部署 JMeter。
- JMeter 的组成部分包含哪些。

完成本章的学习之后，读者就能对 JMeter 的相关概念有一个初步的了解。

# 第 2 章

# 认识性能测试

对 JMeter 有了一个初步认识后，读者还需要对软件性能测试有一个初步认识。性能测试经过长期不断的发展和积累，已经形成了一套完整的测试体系。在这种完备的测试体系下，已经诞生了性能测试工程师这样的专业岗位。本章将完整地介绍性能测试的指标、基本概念以及流程等内容，读者需要掌握这套性能测试体系，并为后面性能测试技能的学习打下基础。

## 2.1 性能的基本概念

### 2.1.1 什么是性能

性能通常可以理解为一个系统实现其功能的能力，也可以理解为在特定的工作负载下能够完成多少任务或处理多少数据的能力。性能可以从宏观和微观两个角度来认识，如图 2-1 所示。

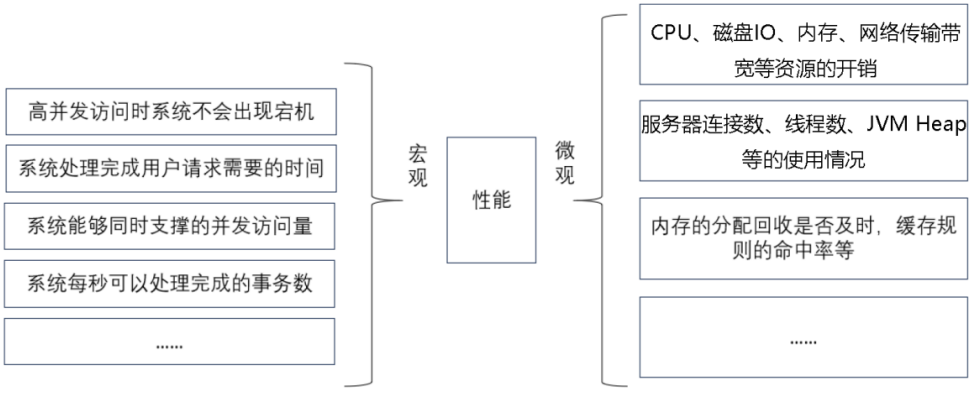


图 2-1　如何理解性能

性能从宏观上可以描述为系统能够稳定运行。其主要指标包括：高并发访问时系统不会出现宕机、系统处理完成用户请求需要的时间、系统能够同时支撑的并发访问量、系统每秒可以处理完成的事务数等。

性能从微观上可以描述为处理每个事务的资源开销。资源的开销可以包括 CPU、磁盘 I/O、内存、网络传输带宽等，甚至可以体现为服务器连接数、线程数、JVM Heap 等指标的使用情况，也可以表现为内存的分配回收是否及时、缓存规则的命中率等。

当然，不同的用户群体对性能的理解可能会存在很大的差异。普通的系统客户可能更加关心系统的响应时间和系统的稳定性：

- 从浏览器中访问的页面，还要让我等多久才能加载出来？
- 为什么有时会访问报错？
- 为什么会提示当前系统使用繁忙，请稍后重试。

架构师可能更加关心架构设计是否合理：

- 应用架构设计是否合理？
- 技术架构设计是否合理？
- 数据架构设计是否合理？
- 部署架构设计是否合理？

软件开发工程师可能更加关心代码编写的性能，代码是否需要优化和调整：

- 代码是否存在性能问题？
- JVM 中是否有不合理的内存分配和使用？
- 线程同步和线程锁是否合理？
- 代码的计算算法是否可以进一步优化，以减少 CPU 的消耗时间？

运维工程师可能更加关心系统的监控以及稳定性情况：

- 服务器各项资源使用率在正常范围内吗？
- 数据库的连接数在正常范围内吗？
- SQL 执行时间正常吗，是否存在慢查询日志？
- 系统能够支撑 7×24 小时连续不间断的业务访问吗？
- 系统是高可用的吗，服务器节点宕机了会影响用户使用吗？
- 对节点扩容后，可以提高系统的性能吗？

测试工程师可能会从专业的性能测试角度来进行考量：

- 系统的平均响应时间达标了吗？
- 系统的 TPS（Transactions Per Second，每秒事务数）是多少？
- 系统的并发用户能支持多少？
- 系统的吞吐量能支持多少？

## 2.1.2　性能测试的意义

### 1. 提升用户体验

一个好的软件系统性能会让用户在访问系统时感到轻松和愉快。用户体验好了后，很可能会成为该系统的回头客，并且会向身边的人推荐该软件系统，从而提高该软件系统的用户转化率。如果一个系统访问很慢，性能不佳，那么很多用户可能会抛弃使用该系统，从而转向竞争对手的软件产品，这样就会造成用户流失。良好的性能体验，可以让软件系统在市场竞争中脱颖而出。

### 2. 降本增效

通过性能测试和性能分析，可以精准地定位出软件系统中的资源瓶颈，避免过度堆积硬件资源来提高性能所造成的成本增加。如果不知道真实的性能瓶颈在哪里，只是增加大量的硬件资源，不但不能有效地提高系统的性能，反而会造成硬件资源的浪费。反之，找到了性能瓶颈所在，不但可以彻底解决性能问题，还可以做到降本增效。

### 3. 辅助业务决策

通过性能测试，可以评估出软件系统的负载承受能力，从而提前制订出应急预案。比如，某电商网站在双 11 大促时，需要做促销或者秒杀活动，如果不提前做性能压测，评估出系统的负载承受能力，那么营销团队在投放广告或者宣传时，就不知道应该在多大范围内来做广告投放和宣传，因为会担心系统的负载承受能力不够。

### 4. 系统风险预防

通过性能测试，可以发现一些在功能测试阶段无法发现的隐藏问题。比如，高并发场景下的系统稳定性，是否会出现系统崩溃、服务器内存泄露导致资源耗尽、数据库死锁等各种问题，造成系统不能提供 7×24 小时的稳定访问服务。

## 2.1.3　常见的性能指标

衡量一个系统性能的好坏，通常会使用一些性能指标来进行分析和描述，以下是一些常用的性能指标。

### 1. 响应时间

响应时间是指请求或者某个操作从发出的时间到收到服务器响应的时间之间的差值。在性能测试中，一般统计的是事务的响应时间。

图 2-2 所示是一次标准 HTTP 请求的处理路径，其响应时间的计算方式就是所有路径消耗的时间和每个服务器节点的处理时间的累加，通常是“网络时间+应用程序的处理时间”。

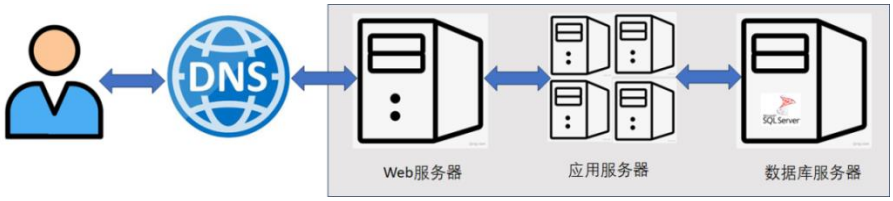


图 2-2 HTTP 请求的处理路径

### 2. TPS/QPS

在性能测试的专业术语中，事务通常指自定义的某个操作或者一组操作的集合。例如，在一个系统的登录页面上，输入用户名和密码，从单击登录按钮开始到登录完成跳转到新的页面，并且新的页面完全加载完成，这一系列操作就可以定义为一个事务。事务通常具有原子性。原子性是指某一个或者某一组操作要么都执行，要么都不执行。在性能测试中，事务的性能指标包括 TPS 和 QPS。

（1）TPS 是 Transaction Per Second 的缩写，即系统每秒能够处理的交易和事务的数量，一般统计的是每秒通过的事务数。

（2）QPS 是 Query Per Second（每秒查询率）的缩写，是对一个特定的查询服务器在规定时间内所处理流量多少的衡量标准。

### 3. 并发用户

在真实的用户操作中，用户的每个相邻操作之间都会有一定的间隔时间（在性能测试中，我们通常会称之为用户的思考时间），所以并发用户一般有绝对并发和相对并发之分。

（1）绝对并发：指某个时间点向服务器发出请求的并发用户数。

（2）相对并发：指一段时间内向服务器发出请求的并发用户总数。

单就性能指标而言，系统的并发用户数是指系统可以同时承载的、正常使用系统功能的用户总数量。

这里针对并发用户进行举例说明，如图 2-3 所示，在京东购物网站上购买一件商品的流程包括登录、浏览商品、把商品加入购物车、去购物车结算、确认商品清单、确认收货地址信息，最后提交订单去支付。如果 200 人同时按照这个流程去购买一件商品，但因为每个人购买商品的速度有快有慢，所以在同一时间点向服务器发出请求的用户肯定不会有 200 个，会远远小于 200 个。我们假设同一时间点向服务器发出请求的用户数为 20，那么 200 就是相对并发用户数，而 20 就是绝对并发用户数。

通常情况下 TPS/QPS、并发数、响应时间三者之间的关系是：TPS/QPS = 并发数 / 平均响应时间。

### 4. PV/UV

PV 和 UV 是衡量 Web 网站用户访问情况的两个非常重要的指标，具体描述如下：

（1）PV：Page View 的简写，即页面的浏览量或者点击量。用户每次对系统或者网站中任何页面的访问均会被记录一次，如果用户对同一页面进行多次访问，那么访问量会进行累加。PV 一般是衡量电子商务网站性能容量的重要指标，PV 的统计可以分为全天 PV、每个小时的 PV 以及峰值 PV（高峰 1 小时的 PV）。

（2）UV：Unique Visitor 的简写，即系统的独立访客。访问网站系统的一台计算机客户端会被称为一个访客，每天 00:00 到次日 00:00 期间的相同客户端只能被计算一次。同样地，UV 的统计也可以分为全天 UV、每个小时的 UV 以及峰值 UV（高峰 1 小时的 UV）。

每秒的 PV 数（PV/s）一般是由 TPS 通过一定的模型转化的，比如，如果把每一个完整的页面都定义为一个事务，那么 TPS 就可以等同于 PV/s。PV 和 UV 之间一般存在一个比例。PV/UV 可以理解为每个用户平均访问的页面数，这个比值在不同的时间点会有所波动。比如，双 11 电商大促时，PV/UV 的值会比平时高很多。

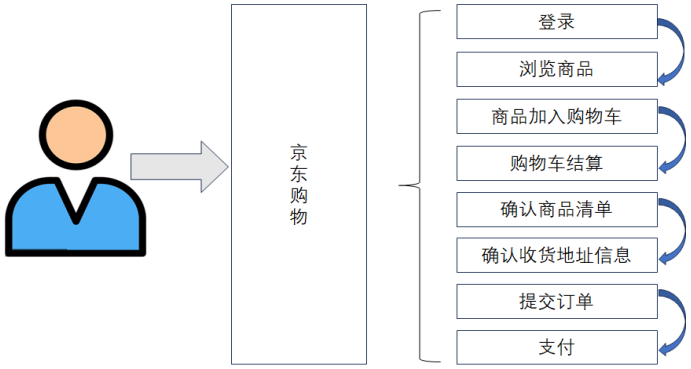


图 2-3　京东购买商品示例

### 5. 点击率

每秒的页面点击数（Hit）称为点击率，如图 2-4 所示。该性能指标反映了客户端每秒向服务端提交的请求数，通常一个 Hit 对应了一次 HTTP 请求。在性能测试中，我们一般不发起静态请求（指的是对静态资源的请求，比如 JS、CSS、图片文件等），所以 Hit 通常指的是动态请求。在性能测试中，我们之所以不发起静态请求，是因为很多静态请求不需要经过应用服务器的处理，要么直接通过 CDN 缓存，要么直接请求到 Web 服务器就被处理完成了。

### 6. 吞吐量

吞吐量是指系统在单位时间内处理客户端请求的数量。一个系统的吞吐量一般与一个请求处理对 CPU、网络带宽、I/O 和内存资源等的消耗紧密相连。从不同的角度来看，吞吐量的计算方式可以不一样。

（1）从业务角度：吞吐量可以用请求数/秒、页面数/秒等来进行衡量。
（2）从网络角度：吞吐量可以用字节/秒来进行衡量。
（3）从应用角度：吞吐量指标反映的是服务器承受的压力，即系统的负载能力。

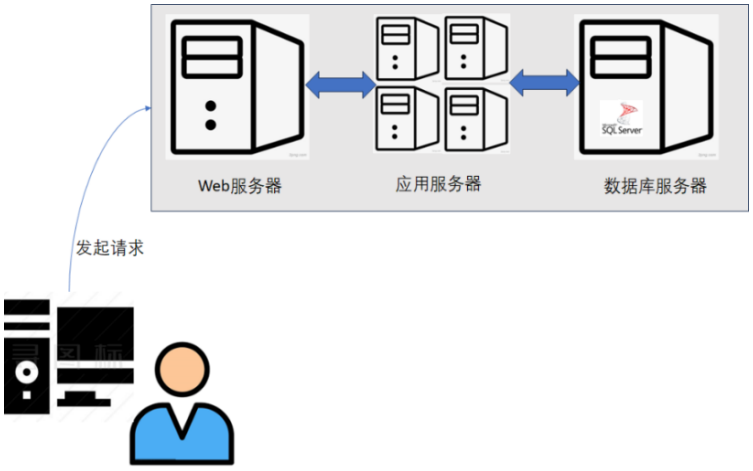


图 2-4　客户端点击率

7. 资源开销

资源开销是指每个请求或者事务对系统资源的消耗，用来衡量请求或者事务对资源的消耗程度。例如，对 CPU 的消耗可以用占用 CPU 的秒数或者核数来衡量，对内存的消耗可以用内存使用率来衡量，对 I/O 的消耗可以用每秒读写磁盘的字节数来衡量。

在性能测试中，资源的开销是一个可以量化的概念，资源开销的情况对性能指标有着重要的影响。我们在做性能优化时，都是尽可能让每一个请求或者事务对系统资源的消耗减少到最小。

## 2.2　性能测试的基本概念

### 2.2.1　性能测试的种类

从软件性能测试的专业角度来看，性能测试通常分为如下几种类别：

（1）性能测试：通常是指统计一个软件系统在正常负载下的各项性能指标，或者通过调整并发用户数，使系统资源的利用率处于正常水平时获取到系统的各项性能指标。

（2）负载测试：通常是指寻找系统在不同负载下的性能表现，通过负载测试可以知道系统在不同负载下性能变化的过程，从而寻求到性能的拐点。例如，在负载测试时，通过不断递增并发用户数，可以观察各项性能指标的变化规律，找到系统能达到的最大 TPS，并且观察此时系统处理的平均响应时间和各项系统资源的消耗情况。

（3）压力测试：系统在高负载下的性能表现。该项测试主要为了寻求系统能够承受的最大负载以及此时的吞吐率。通过该测试也可以发现系统在超高负载下是否会出现崩溃而无法访问，以及在负载减小后，系统性能能否自动恢复。

（4）基准测试：针对待测系统进行版本执行的测试，采集各项性能指标作为后期版本性能的对比。

（5）稳定性测试：以正常负载或者略高于正常负载来对系统进行长时间的测试，检测系统是否可以长久稳定地运行，以及系统的各项性能指标会不会随着时间发生明显变化。

（6）扩展性测试：通常用于新上线的系统或者新搭建的系统环境。先测试单台服务器的处理能力，然后慢慢增加服务器的数量，测试集群环境下单台服务器的处理能力是否有损耗，集群环境的处理能力是否可以呈现稳定增加。

### 2.2.2　性能测试的场景

性能测试的场景类型通常包含如下几种：

（1）业务场景：通常指的是系统的业务处理流程，描述具体的用户行为，通过对用户行为进行分析，划分出不同的业务场景。业务场景是性能测试时测试场景设计的重要来源。

（2）测试场景：测试场景是对业务场景的真实模拟，测试场景的设计应该尽可能贴近真实的业务场景。有时由于测试条件的限制，也可以适当作一些调整和特殊的设置。

（3）单个场景：指的是只涉及单个业务流程的测试场景，目的是测试系统的单个业务处理能力是否达到预期，并且得到系统资源利用正常情况下的最大 TPS、平均响应时间等性能指标。

（4）混合场景：测试场景中涉及多个业务流程，并且每个业务流程在混合的业务流程中所占的比重存在不同，该比重一般根据实际的业务流程来设定，尽可能符合实际业务的需要。该测试场景的目的是测试系统的混合业务处理能力是否满足预期要求，并且评估系统的混合业务处理容量最大能达到多少。

## 2.3　性能测试的流程

通常情况下，性能测试一般会经历如图 2-5 所示的阶段。这些阶段可以和很多性能测试工具对应起来，比如分析性能测试结果可以结合 JMeter 中的监听器来辅助测试。

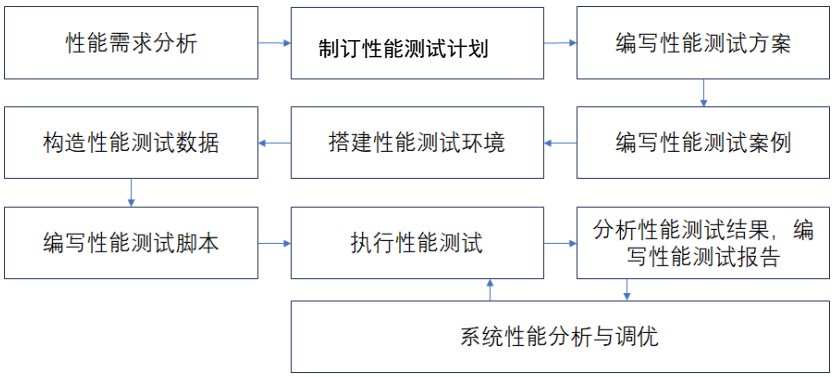


图 2-5　性能测试流程

### 2.3.1 性能需求分析

（1）熟悉被压测系统的基本业务流程，明确此次性能测试要达到的目标，与产品经理、业务人员、架构师、技术经理一起沟通，找到业务需求的性能点。

（2）熟悉系统的应用架构、技术架构、数据架构、部署架构等，找到与其他系统的交互流程，明确系统部署的硬件配置信息与软件配置信息，把对性能测试有重要影响的关键点明确地列举出来。一般这些关键点包括：

- 用户发起请求的顺序、请求之间的相互调用关系。
- 业务数据流走向、数据是如何流转的、经过了哪些应用服务、经过了哪些存储服务，并评估被压测系统可能存在的重点资源消耗，是 I/O 消耗型、CPU 消耗型还是内存消耗型，这样在执行压测时可以重点进行监控。
- 关注应用的部署架构。如果是集群部署，那么压测时需要关注应用的负载均衡转发是否均匀，每台应用服务器资源消耗是否大体一致。
- 和技术经理一起沟通，明确应用的并发架构采用的是多线程处理还是多进程处理，重点关注是否会死锁、数据是否不一致、线程同步锁是否合理（锁的粒度一般不宜过大，若过大，则可能会影响并发线程的处理）等。

（3）明确系统上线后可能会达到的最大并发用户数、用户期望的平均响应时间以及峰值时的业务吞吐量，将这些信息转换为性能需求指标。

### 2.3.2 制定性能测试计划

性能测试计划是性能测试的指导，也是一系列测试活动的依据。在制订性能测试计划时，需要明确系统的上线时间点、当前项目的进度以及所处的阶段、可供调配的硬件资源和性能测试人员。一个完整的性能测试计划一般包括如下几个部分。

#### 1. 性能测试计划编写的目的

性能测试计划主要是作为整个性能测试过程的指导，让性能测试环境的搭建、测试策略的选取、任务与进度事项的跟踪、性能测试的风险分析等事项有序地进行，同时也需要明确此次性能测试预期要达到的标准以及性能测试完成后退出测试的条件。

#### 2. 明确各个阶段的具体执行时间点以及对应的责任人

（1）预计由谁何时开始性能需求分析，何时结束性能需求分析。
（2）预计由谁何时完成性能测试方案的编写，何时结束性能测试方案的编写。
（3）预计由谁何时完成性能测试案例的编写，何时结束性能测试案例的编写。
（4）预计由谁何时开始搭建性能测试环境，何时结束性能测试环境的搭建。
（5）预计由谁何时开始准备性能测试需要的数据，何时准备完毕。
（6）预计由谁何时开始编写性能测试脚本，何时编写完毕。

（7）预计由谁何时开始性能测试的执行，何时完成性能测试的执行。

**3. 性能测试风险的分析和控制**

性能测试风险的分析和控制主要是评估可能存在的风险和不可控的因素，以及这些风险和因素对性能测试可能产生的影响，并针对这些风险因素给出对应的短期和长期的解决方案。性能测试风险一般包括如下 3 个方面：

1）性能测试环境因素

如果无法按期完成性能测试环境的搭建，那么这中间的问题既可能是硬件引起的，也可能是软件引起的。硬件问题一般包括性能压测服务器无法按时到位、服务器硬件配置无法满足预期（一般要求性能压测服务器硬件配置等同于生产环境，服务器的节点数可以少于生产环境，但是需要保证每个应用服务至少部署了两台节点服务器）。软件问题可能包括性能测试环境软件配置无法和生产环境保持一致（一般要求性能压测环境软件配置，比如软件版本、数据库版本、驱动版本等要和生产环境完全一致）。

2）性能测试人员因素

如果性能测试人员无法按时到位参与项目的性能测试，肯定会导致性能测试无法按期进行，需要立即向项目经理汇报，以确保可以协调到合适的人员。这是一个非常严重的风险。

3）性能测试结果无法达到预期

即系统的性能无法达到生产预期上线要求或者存在性能问题无法解决。性能调优本身其实就是一个长期不断优化的过程，此时可以看看能否通过服务器的横向或者纵向扩容来解决。如果通过服务器扩展还是无法解决问题，那么需要提前上报风险。

## 2.3.3　编写性能测试方案

在有了性能测试计划后，我们就需要按照性能需求分析的结果来制订性能测试方案，即按照什么样的思路和策略去测试、需要设计哪些测试场景、测试场景执行的先后顺序、每个测试场景需要重点关注的性能点等。一般编写性能测试方案包括如下工作。

**1. 设计测试场景**

（1）单一场景设计：单一业务流程的处理模式设计。

（2）混合场景设计：多个业务流程的混合处理模式设计。

**2. 定义事务**

在测试方案中需要明确定义压测事务，以方便分析响应时间（特别是在混合场景中，定义压测事务可以方便地分析每一个场景响应时间的消耗）。比如，我们对淘宝商城购买商品这一场景进行压测，可以把提交订单定义为一个事务，把支付定义为一个事务，还可以把提交订单加支付定义为一个事务，如图 2-6 所示。事务的定义取决于性能压测时自己关注的业务流程。在压测结果中，如果响应时间较长，就可以对每一个事务进行分析，确定哪个事务耗时最长。

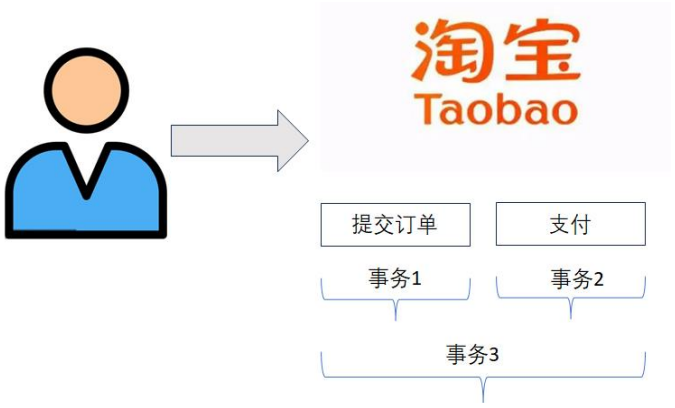


图 2-6　淘宝商城购买商品示例

### 3. 明确监控对象

针对每个场景，明确可能的性能瓶颈点（比如数据库查询、Web 服务器服务转发、应用服务器等）和需要监控的对象（比如 TPS、平均响应时间、点击率、并发连接数、CPU、内存、I/O 消耗等）。

### 4. 定义测试策略

明确性能测试的类型。确定需要进行哪些类型的性能测试，比如负载测试、压力测试、稳定性测试等。

明确性能测试场景的执行顺序，一般先执行单场景测试，后执行混合场景测试。

如果是进行压力测试，还需要明确加压的方式。比如，按照开始前 5 分钟增加 20 个用户，然后每隔 5 分钟增加 20 个用户来进行加压，如图 2-7 所示。

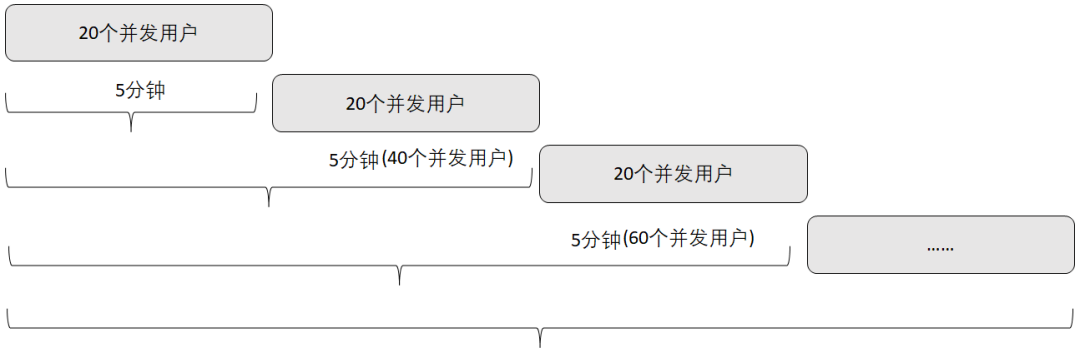


图 2-7　压力测试加压示例

### 5. 性能测试工具的选取

性能测试工具有很多，常见的有 JMeter、Loadrunner、nGrinder 等，那么如何选取合适的性能测试工具呢？可以考虑以下因素：

（1）一般性能测试工具都是基于网络协议开发的，所以我们需要明确待压测系统使用的协

议，尽可能和待压测系统的协议保持一致或者至少要支持待压测系统的协议。

（2）理解每种工具的实现原理，比如哪些工具适用于同步请求的压测，哪些工具适用于异步请求的压测。

（3）压测时明确连接的类型，比如属于长连接还是短连接，连接多久能释放。

（4）明确性能测试工具并发加压的方式，比如是多线程加压还是多进程加压。通常默认采用的是多线程加压。

**6. 明确硬件配置和软件配置**

（1）硬件配置一般包括：服务器的 CPU 配置、内存配置、硬盘存储配置、集群环境下集群节点的数量配置等。

（2）软件配置一般包括：

- 操作系统配置：操作系统的版本以及参数配置需要同线上保持一致。
- 应用版本配置：应用版本要和线上保持一致，特别是中间件、数据库组件等的版本。因为不同的应用版本，其性能可能不一样。
- 参数配置：比如 Web 中间件服务器的负载均衡、反向代理参数配置、数据库服务器参数配置等。

（3）网络配置：一般为了排除网络瓶颈，除非有特殊要求，建议在局域网下进行性能测试，并且明确压测服务器的网卡类型以及网络交换机的类型，比如是否属于千兆网卡，以及交换机属于百兆交换机还是千兆交换机等。这对我们以后分析性能瓶颈会有很大的帮助，在网络吞吐量较大的待压测系统中，网络有时也很容易成为一个性能瓶颈。

## 2.3.4　编写性能测试案例

性能测试案例是对性能测试方案中性能压测场景的进一步细化，其内容一般包括如下 3 点：

（1）预置条件：一般指执行此案例需要满足何种条件，性能测试案例才可以执行。比如，性能测试数据需要准备到位、性能测试环境需要启动成功等。

（2）执行步骤：详细描述案例的执行步骤。一般需要描述测试脚本的录制和编写、脚本的调试、脚本的执行过程（比如，如何加压、每个加压的过程持续多久等）、观察和记录的性能指标、性能曲线的走势、监控哪些性能指标等。

（3）性能预期结果：描述性能测试预期需要达到的结果，比如 TPS 需要达到多少，平均响应时间需要控制在多少以内，服务器资源的消耗需要控制在多少以内等。

## 2.3.5　搭建性能测试环境

性能测试环境的搭建需要注意如下几点：

（1）尽可能与实际生产环境的配置保持一致，不可减少其中的相关组件，实际生产环境中

的某些组件在性能测试环境中必须部署。

（2）一般生产环境的服务器数量和配置都很高，但在性能测试环境，不可能以这么高的成本去部署完全一样的硬件机器。性能测试环境上的机器数量和机器配置可以按照一定比例进行缩减，但如果是分布式系统，那么服务的机器节点需要两个或者两个以上，并且节点资源配置不能太低。

（3）操作系统版本以及在操作系统上部署的相关软件（比如中间件软件、应用容器软件等）版本、数据库软件版本必须与实际生产环境完全一致。网络环境必须与实际生产环境保持一致，因为网络是一个容易成为性能瓶颈的地方。

### 2.3.6 构造性能测试数据

性能测试的结果是否准确，在一定程度上还取决于测试数据的数量和质量。

如果不是使用实际的生产环境进行线上的性能测试（比如淘宝、天猫、苏宁易购等网站可以在双 11 大促前，允许在特定的时间段进行生产环境的压力测试），那么在性能测试环境中，就需要提前导入能够模拟生产环境的基础数据，比如用户数据、角色权限数据等，并且基础数据需要有足够的量，或者尽可能和生产环境一致，或者只略低于生产环境。

测试场景和测试案例中需要的数据，可以通过从生产环境中拉取数据并脱敏后导入，或者使用数据脚本进行批量构造。数据量尽可能和生产环境的数据库中的数据量在一个数量级上，或者根据性能测试环境硬件配置和生产环境硬件配置的比例来评估测试数据和生产数据的量级比例为多少是合适的。

数据的质量也非常重要，比如一个查询操作的性能测试场景，如果构造的数据都是查询参数无法查询到的数据，那么这个性能测试的结果肯定不好。

### 2.3.7 编写性能测试脚本

性能测试脚本主要包含如下内容：

（1）按照性能测试场景，开始录制性能测试脚本或者直接编写性能测试脚本，此时可能用到的常见性能测试工具包括 JMeter、Loadrunner、BadBoy、nGrinder 等。

（2）根据准备好的测试数据，对性能测试脚本进行参数化，添加集合点、事务分析点等。

（3）对性能测试脚本进行试运行调试，确保不出现报错，并且可以覆盖测试场景中的所有操作。

### 2.3.8 执行性能测试

性能测试的执行过程如图 2-8 所示。

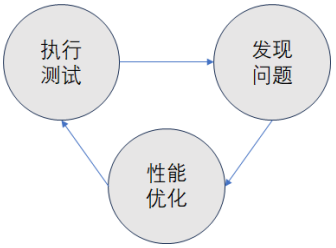


图 2-8　性能测试的执行过程

（1）执行测试：完成每一个性能测试场景和案例的执行，记录相关的性能测试结果，明确性能曲线的变化趋势，获取性能的拐点等。

（2）发现问题：根据性能测试的结果，评估性能数据是否可以满足预期，从性能测试结果数据中分析存在的性能问题。

（3）性能优化：针对性能问题，进行性能定位和优化，然后进行二次压测，直至性能数据可以满足预期，性能测试问题得到解决。

### 2.3.9　编写性能测试报告

性能测试报告是性能测试的主要产出物之一，也是对性能测试结果和测试数据的总结与分析。它记录了系统在不同负载和场景下的性能表现、发现的性能问题、性能测试的结论和性能调优的建议等。其详细内容说明如下。

（1）测试环境描述：描述进行性能测试时使用的实际环境，通常需要包括：

- 硬件信息：比如服务器的型号、CPU 的核数、内存的大小等常见的信息。
- 软件系统信息：比如操作系统的类型及其版本、数据库的类型及其版本等常见的软件信息。
- 网络配置信息：比如网卡的型号、带宽大小等常见的网络传输方面的信息。

（2）测试案例和场景：描述执行的测试场景和案例，以及对应的测试场景和案例的测试结果是否达到了预期结果。

（3）测试结果记录：详细记录每个测试场景和案例执行完成后得到的详细结果数据。通常包括并发数、吞吐量、TPS、响应时间、CPU 和内存以及网络资源的使用率等。

（4）测试结果分析：对每个测试场景和案例的测试结果做深入的性能问题分析，挖掘出可能的性能瓶颈和问题等。

（5）测试问题与缺陷记录：整理出整个性能测试过程中发现的问题和缺陷，以及这些问题和缺陷造成的影响。

（6）性能调优建议：基于性能测试的结果、分析验证或者凭借性能测试的过往经验，给出当前测试系统的性能改进建议。

（7）性能测试结论：基于以上分析，给出最终性能测试是否能够达到预期的结论，以及可

能存在的性能风险等信息。

## 2.4 本章总结

本章主要介绍了性能测试的基本概念、常见的性能指标、性能测试的流程等。读者需要掌握以下重点内容：

- 性能测试的基本概念，包括什么是性能、性能测试的种类、性能测试的场景等。
- 常见的性能指标，包括响应时间、TPS/QPS、并发用户、并发用户的数量、PV/UV、点击率、吞吐量和资源开销等。
- 性能测试的流程，包括如何进行性能需求分析，怎么去制订性能测试计划，如何设计出一个好的性能测试方案，如何编写对应的测试案例，如何搭建测试环境和构造性能测试数据，如何编写性能测试脚本，如何执行性能测试，以及如何编写最终的性能测试报告。

通过学习本章内容，读者应该能够对性能测试有一个基本的认识，在接到一个性能测试任务时，知道如何按照性能测试流程做好相应的准备工作。